Hierarchical Routing in Dynamic α-Doubling Networks

Dominique Tschopp, Suhas Diggavi, Matthias Grossglauser

Abstract

In this paper we formulate the problem of routing over dynamic networks with finite doubling dimension. This is motivated by communication in mobile wireless networks, where the communication graph topology changes over time, but has some geometric properties, motivating the model for finite doubling dimension. Since wireless network bandwidth is precious, we consider communication cost required to set up the routing on such dynamic networks. We show that under appropriate modeling on time-changes of the dynamic network, we can build addressing with small total overhead and maintain routing with constant stretch for dynamic doubling metric networks.

School of Computer and Communication Sciences, EPFL, {dominique.tschopp,suhas.diggavi,matthias.grossglauser}@epfl.ch

I. Introduction

Communication networks are networks of computing devices such as computers, cell phones or sensors. Those devices can interact through communication channels, which can be wired or wireless. We can naturally define a communication graph $\mathcal{G}$ by associating a set of vertices V to the set of computing devices and a set of edges E to the set of communication channels *i.e.,* we add an edge between two vertices if the corresponding devices can interact directly over a communication channel. We will use the terms vertices and nodes interchangeably in the sequel. Note that these edges could be weighted to reflect a characteristic of the channel. Possible channel characteristics include capacity, delay and reliability. In most realistic scenarios, the communication graphs are not complete and nodes have to communicate by using intermediate nodes in $\mathcal{G}$ as relays.

Informally, the problem of routing can be thought of as the problem of finding a path in $\mathcal{G}$ between two nodes which wish to communicate together. Computing devices typically have limited capabilities, or are limited by their environment, which must be taken into account when designing a routing scheme. Battery powered nodes, for instance, should communicate in an energy efficient way to increase the lifetime of the computing device. In a wireless communication network, the number of nearby nodes which can communicate simultaneously is limited because of interferences on the wireless channel. In sensor networks, memory might also be a limiting factor. These limitations lead to three commonly used cost functions to measure the performance of a routing scheme:

(i) *Control traffic overhead* $[bits]$: how many bits need to be transmitted in the network in order to establish or maintain routes
(ii) *Memory* $[bits]$: how many bits do nodes need to store in order to allow routing
(iii) *Computational complexity*: How much computational power does the execution of the routing algorithm require
(iv) *Stretch*: How good are the routes computed with the routing scheme compared to the best possible paths.

Note that these costs come in different flavors. One could for example work with either average or worst case cost, and require that a route exists between all pairs of nodes at any time or only when two nodes actually communicate.

In this paper, we focus only on transmission overhead as the critical resource. There is an obvious tradeoff between this transmission overhead, and the quality of routes that are available for messages to be sent through the network. On one extreme, we can compute full routing tables, which is very costly, and have shortest paths available at all times between any pair of nodes. At the other extreme, we could avoid sending any control information, and let messages travel randomly through the network until encountering the destination. An important question is the achievable tradeoff between these two extremes, and in particular, whether we can significantly reduce the control overhead while sacrificing only little efficiency. In other words, while consuming as little bandwidth as possible we want to guarantee as low a stretch for all pairs of nodes at any time as possible. This is in contrast to an existing body of work that focuses on memory as the critical resource (see [1] and references therein).

Another novel aspect of our work is the study of dynamic communication networks *i.e.,* networks in which the topology evolves over time. Our main motivation for investigating routing in such dynamic environments is the study of wireless multi-hop networks and hybrid wired and wireless networks. In this type of networks nodes are usually mobile and have a limited communication range, leading to a dynamic communication topology. Further, the inherently random nature of the wireless channel inevitably leads to the permanent disappearance and appearance of new communication links. Hence, we extend the communication graph model to make it time dependent *i.e.,* we model the network as a time-varying graph $\mathcal{G}^{(t)}(V, E)$. For simplicity t is discrete and the graph is unweighted. The latter assumption is reasonable if all nodes have similar characteristics. Note that we do not make any particular assumptions on the capacity of the communication links. Our focus is on the total amount of information transmitted in the network and we allow some nodes to be temporarily overloaded. One can immediately see that these dynamics introduce new challenges for the design of routing protocols. Indeed, routes cannot be established one time for all anymore. Rather, routes must be permanently updated to guarantee low stretch routing.

However, we do not assume that $\mathcal{G}^{(t)}(V, E)$ is a general graph; rather, the graph reflects the underlying geometry of wireless networks. Specifically, the nodes of a wireless network are embedded in a two or three-dimensional Euclidean space. Whether two nodes are directly connected or not depends strongly on the Euclidean distance between these nodes, despite the channel uncertainty due to fading. Nodes which are not connected directly and wish to communicate with each other must use other nodes as relays to communicate over multiple hops. In a sufficiently dense network, the distances in the connectivity graph reflect the underlying geometry of the Euclidean

space. This leads us to expect that the graph is inherently low-dimensional *i.e.,* that it is embeddable with constant distortion in a low-dimensional Euclidean space. One consequence of this is that the number of nodes reachable from a starting node v only grows polynomially with some exponent α, which is a key property we exploit here. This is captured in the assumption that the metric space defined by the graph $\mathcal{G}^{(t)}$(V,E) has a constant doubling dimension α (see definition in Section II). In the Appendix, we give some additional details about the α-doubling nature of Unit Disk Graphs and Random Geometric Graphs. Another property of wireless networks is that although channel fading may result in individual links appearing and disappearing, if the network is sufficiently dense, then there is a set of alternate paths of similar lengths between any two nodes; therefore, the distance function does not fluctuate too much in a short time interval. Also, topology changes due to node mobility are constrained by the speed at which nodes can move around. Hence, nodes positioned very far apart *e.g.,* laptop users on opposite sides of a town, are very unlikely to suddenly get a direct communication channel. Similarly, nodes with a direct communication channel might get disconnected but it is likely that there exists a short path in the communication graph between them. We capture this property by constraining the evolution of the communication graph (see definition of κ-constrained metric sequence in Section II).

Routing scheme can be further subdivided into "named" and "addressed" routing schemes (alternatively called "unlabeled" and "labeled" routing schemes, respectively). In named routing, forwarding of packets is based uniquely on the original identifiers (names) of the nodes, which cannot be chosen by the routing algorithm designer. On the other hand, in addressed routing, one can assign addresses to nodes which facilitate packet forwarding. The assumption is usually implicitly made that the source node can learn the address which has been given to the destination at no cost and include it in the packet header. Finally, there also exist hybrid schemes in which the source node can first discover the destination's address using only its name and then route using this address. In wireless routing, when nodes know their geographical coordinates (*e.g.,* obtained using a GPS), they can use these coordinates as addresses for routing. First, however, they need to learn the coordinates of the destination. This is typically achieved by using a "location service", which is a kind of distributed database in the network. This database needs to be updates by nodes with their positions and can be queried by nodes wishing to communicate. This paradigm can also be applied when the addresses of the nodes do not correspond to their geographical coordinates but rather are assigned by the routing algorithm.

In this paper, we first propose a beaconing scheme to address nodes in a static environment. The addresses allow $O(1)$ stretch routing. Then, we show that we can maintain such an addressing in a dynamic environment while not giving up on the stretch guarantees. Finally, we present a named routing scheme which relies on the addressing scheme to decompose the communication graph but never explicitly uses the addresses for packet forwarding.

A. Related work

Routing in wireless networks has been a rich area of enquiry over the past decade or more. Two schemes that utilize the underlying geometry of graphs in *static* wireless networks algorithms are the works presented in [2] and the beacon vector routing (BVR) introduced in [3]. Both these schemes are heuristic which build a virtual coordinate system over which routing takes place. They were shown to work well through numerics. However, they utilize an external addressing scheme, to make a correspondence between addresses and names. In [4], we initiated a study on routing for dynamic networks using a virtual coordinate system. For large scale dynamic wireless networks, these heuristics pointed to significant advantages to using some geometric properties for routing and addressing. These results motivated the questions studied in this paper.

There has been a vast amount of research on efficient routing schemes in wired networks (see for example [1]). Most of this work has been focused on the trade-off of memory (routing table size) and routing stretch. There are two main variants of the routing schemes (i) *labeled* (or *addressed*) routing schemes, where the nodes can be addressed so as to reflect topological information. (ii) *Named* routing uses arbitrary node names, and as part of the routing, the location (or address) of the destination needs to be obtained. This addresses the important question of how the node addresses need to be published in the network. Routing in graphs with finite doubling dimension has been of recent interest (see [5], and references therein). In particular [6] showed that one could get constant stretch routing with small routing table sizes for doubling metric spaces, when we use labeled routing. This result was improved to make routing table sizes smaller in [7]. The problem of named routing over graphs with small doubling dimension has been studied in [5] and [8], and references therein. To the best of our knowledge, there has been no prior work on *dynamic* graphs over doubling metric spaces and on control traffic overhead.

Symbol	*Definition*
$\mathcal{G}^{(t)}(V,E)$	Time varying communication graph with vertex set V and edge set E
$\mathcal{N}_G(v)$	The set of direct neighbors of a node $v \in V$ in G
$D(u,v)$	Shortest path distance between nodes u and v
$B(v,r)$	$\{y \in V : D(v,y) \leq r\}$
Δ	$max_{(u,v)\in V^2} D(u,v)$
α_X	The *doubling constant* α_X of a metric space (X,d)
r-cover	$\{Y \subset X : \forall x \in X \exists y \in Y$ s.t. $D(y,x) \leq r\}$
$\mathcal{T}_G$	The control traffic is the total number of control bits transmitted in the network
$route(u,v)$	Sequence of nodes traversed by a packet from u to v
$r(u,v)$	Length of $route(u,v)$
S_{uv}	$\frac{r(u,v)}{D(u,v)}$
$\mathcal{T}$	average control traffic over a sequence of graphs

TABLE I

TABLE OF NOTATIONS

It is worth pointing out that there is no direct correspondence between control traffic and memory. Bounds on memory do not take into account the amount of information which needs to be sent around in the network in order to build routing tables. A good illustration is the computation of the shortest path between two nodes u and v in a graph. While it is sufficient for every node on the path between these two nodes to have one entry for v (of roughly $\log n$ bits *i.e.,* the name of the next hop), computing that shortest path requires a breadth first search of the communication graph and leads to a control traffic overhead of $O(n \log n)$ bits.

B. Organization

In Section II, we define the problem and the notation used in the paper. The main results are briefly stated in Section III. We give the addressed routing schemes in dynamic doubling networks in Section IV. We give some preliminary results on name distribution schemes for named routing for dynamic networks in Section V.

II. PROBLEM STATEMENT AND DEFINITIONS

In this section we define notions necessary to formally state our problems and results. The definitions and notations are summarized in Table I.

We model a communication network as a time dependent unweighted graph $\mathcal{G}^{(t)}(V,E)$ with $|V| = n$, in which we associate vertices to nodes and add an edge between vertices if the corresponding nodes can communicate directly in the underlying network. All graphs in this paper are undirected. For simplicity, t is discrete and takes values in $\mathbb{N}$. We denote by $\mathcal{N}_G(v)$ the set of direct neighbors of a node $v \in V$ in G. We let $D_G(u,v)$ denote the length of the shortest path between nodes u and v in G. Note that (V,D) is a metric space, which we require to have at least two points. The *ball* $B(v,r) = \{y \in V : D(v,y) \leq r\}$ is the set of points at distance at most r from v. We will omit the graph in the subscript whenever it is obvious from the context. We define the diameter of the network as $\Delta = max_{(u,v)\in V^2} D_G(u,v)$[1]. We make the assumption that the shortest distance is 1.

The *doubling constant* α_X of a metric space (X,d) is the smallest value α such that every ball in X *can be covered* by at most α balls of half the radius. The doubling dimension of X is defined as dim(X)=$log_2(\alpha_X)$. A metric is called *doubling* if its doubling dimension is a constant. Finally, and r-cover is a subset of X which is defined as follows:

Definition 1 (r-cover): An *r-cover* of a metric space (X,d) is a subset of nodes $Y \subset X$ such that $\forall x \in X$ there exists a $y \in Y$ with $D(y,x) \leq r$.

In a straightforward manner, one can now define a hierarchical (r,c)-cover

Definition 2 (hierarchical (r,c)-cover): A *hierarchical (r,c)-cover* of a metric space (X,d) is a set of r_j-covers *i.e.,* a set of sets Y_j, such that $r_0 = \Delta$, $r_{j+1} = r_j/c$ and $r_{log_c\Delta} = 1$, for $j \in [log_c\Delta]$.

In the next subsection, we will now define our communication model.

[1]We define V^2 as $V \times V$

A. Communication Model

Since we focus on communications in a wireless network, we are making a few basic assumptions about the communication model. Even though the node transmissions interfere, we assume that we can set up point-to-point wireless links between nodes represented by the edges in the graph. However, we can utilize the broadcast nature of the wireless channel as follows:

Definition 3 (Broadcast Channel Model): Let $G(V, E)$ be an undirected unweighted graph. In the broadcast model, any bit sent out by a node $v \in V$ can be overheard simultaneously by all nodes $u \in \mathcal{N}_G(v)$.
Therefore, with one transmission, we can potentially reach all neighbors in a one-hop neighborhood[2]. We distinguish between *data traffic* and *control traffic*. The former comprises all the packets containing application data. The latter includes all the packets necessary to establish and maintain routes in the network. We mainly use the broadcast property to disseminate control traffic, but the data traffic is routed over paths in the graph, *i.e.,* only the intended next-hop (receiver) listens to the transmission.

Definition 4 (Control Traffic Overhead): The control traffic overhead $\mathcal{T}_G$ of a graph G is the total number of bits of control traffic *transmitted* by all the nodes in the graph.
Formally, a *route* $route(u, v) = \{u, x_1, x_2, ..., v\}$ is sequence of nodes $\in V$ a packet traverses between node u and node v. We denote the length of $route(u, v)$ by $r(u, v) = |route(u, v)| - 1$. Note that we do not count the last hop (the destination itself) as we are interested in the number of nodes which transmit.

Definition 5 (Route Stretch): The stretch S_{uv} of a route $route(u, v)$ is defined as the ratio between the length of the route between u and v and the shortest path $D(u, v)$ *i.e.,* $S_{uv} = \frac{r(u,v)}{D(u,v)}$. We define the (maximum) stretch as $\mathcal{S}_G = \max_{u,v} S_{uv}$.

B. Dynamic Topologies

One of the contributions of this paper is a formulation of the routing problem on dynamic communication networks with communication costs. A dynamic communication network is a network in which the communication topology *i.e.,* the underlying communication graph can evolve over time. The modification of the topology can be due to the appearance of new communication links between nodes which were disconnected beforehand or to communication links which break down. In a wireless network, for instance, the changes in the topology can result from node mobility as well as from the random nature of the wireless channel. We are focusing on networks in which the dynamics are constrained, so that shortest path distances cannot change too quickly over time. This model is natural for networks in which physical constraints limit the changes in topology. The model assumes that network connectivity is dense/homogeneous enough that the changes do not disconnect the network or have critical bottlenecks. In a wireless network, the speed of the nodes is limited, as well as their radio range. Hence, we can only move from one topology to a limited number of new topologies. Also, if the node density is large enough, then there are alternate paths even when some links fail or change. To capture those limits on the topology changes, we model the dynamics of a communication network by looking at a sequence of communication graphs. More formally, we constrain the evolution of the topology in the following way:

Definition 6 (κ-constrained metric sequence): Let $\mathcal{G}^{(j)} = (X, D)^{(j)}$, $j \in \mathbb{N}$, be a sequence of α-doubling graph metric spaces on X. Let $D^{(m)}(i, j)$ denote the distance between i and j in $(X, D)^{(m)}$. We say that that the sequence $\mathcal{G}^{(1)}, \mathcal{G}^{(2)}, ...$ is κ-constrained if $max_{(i,j)\in X} |D^{(m)}(i, j) - D^{(m+1)}(i, j)| < \kappa$, $\forall m$.
Note that we have interchangeably used the terminology of the sequence of finite metric space and the graph sequence. Our focus in this paper is the long-term communication cost and routing stretch over *each* graph in the sequence of dynamic topologies defined by κ-constrained metric sequences. More formally, we require to characterize the average communication cost and worst-case routing stretch as follows.

Definition 7 (Average Control Traffic Overhead): The average control traffic overhead $\mathcal{T}$ of a sequence of graphs $G^{(1)}, G^{(2)}$ is the average number of bits (over time) of control traffic *transmitted* by all the nodes in the graph given by $\mathcal{T} = \lim_{T\to\infty} \frac{1}{T} \sum_{j=1}^{N} \mathcal{T}_{G^{(j)}}$.

[2]If the degree of each node is bounded by a constant. then this broadcast property changes the communication cost only by a multiplicative constant.

For the dynamic graph we define the routing stretch $\mathcal{S}$ as the *worst* stretch over the sequence of graphs, *i.e.*, $\mathcal{S} = \max_{j \in \mathbb{N}} \mathcal{S}_{G^{(j)}}$. Finally, in our bounds for dynamic communication graphs, the diameter Δ of the network corresponds to the maximum diameter over the sequence of graphs *i.e.*, $\Delta = max_t max_{(u,v) \in V^2} D_{\mathcal{G}^{(t)}}(u, v)$.

C. Problem Statement

We first seek an addressed routing scheme for static α-doubling unweighted communication graphs with the broadcast channel model which leads to a small control traffic overhead. We then want to extend this scheme to κ-constrained sequences of such graphs. In this second step, we want to utilize the properties of the communication graph to develop a scheme which results in a low average control traffic overhead. Finally, we want to develop a named routing scheme applicable to both static and dynamic environments. Again, we aim at minimizing the additional overhead for name distribution.

III. Main Results

Our main results are summarized in three theorems. The first theorems gives a bound on the control traffic for addressed routing in a static graph. The second theorem gives an upper bound for the average control traffic for addressed routing in dynamic graphs. Finally, the third theorem gives an upper bound for the average control traffic overhead for named routing. For all three routing schemes, we require the stretch of the routes to be $O(1)$.

Theorem 1 (Addressed routing in static networks): Let $\mathcal{G}(V, E)$ be an α-doubling unweighted graph of diameter Δ. Under the broadcast channel model, there exists an addressing scheme with $O(n \log \Delta \log n)$ bits of control traffic overhead which allows $O(1)$ stretch routing for all source destination pairs

Theorem 2 (Addressed routing in dynamic networks): Let $\mathcal{G}^{(t)}(V, E)$, with $t \in \{1, \ldots, T\}$ be a κ-constrained α-doubling unweighted graph sequence, where κ is a constant. Under the broadcast channel model, there exists an addressing scheme with $O(nT \log n)$ bits of overhead, *i.e.*, $\mathcal{T} = O(n \log n)$ of average control traffic overhead which allows $\mathcal{S} = O(1)$ stretch routing for all source destination pairs.

Theorem 3 (Routing in Dynamic Networks): Let $\mathcal{G}^{(t)}(V, E)$, $t < T$ be a κ-constrained α-doubling unweighted graph sequence, where κ is a constant. Under the broadcast channel model, there exists a named routing scheme with $O(nT \log n \log \Delta)$ bits of control traffic overhead over a period of time T, *i.e.*, $\mathcal{T} = O(n \log n \log \Delta)$, which allows $O(1)$ stretch routing for all source destination pairs.

In the next sections, we will present practical algorithms which achieve those bounds. Our routing scheme relies on a beaconing procedure to address the nodes and on a greedy forwarding algorithm

IV. Addressed Routing

In the first part of this section, we will present our addressing scheme for a static environment and prove that it allows $O(1)$ stretch addressed routing. In the second part, we will extend this addressing scheme to dynamic environments.

A. Addressed Routing in Static Graphs

The address of a node can in fact be represented as a routing table as shown in Table II. The address itself consists of a list of node names, which we shall call beacon nodes in the sequel, as well as the distance to these beacons in hops and the level of the beacon in the hierarchy (more details in this section). Additionally, we also store the next hop to the beacon *i.e.*, the name of the neighbor on the route to the beacon. In order to fill the

beacon name	*distance* $[hops]$	*level*	*next hop*
$\vdots$	$\vdots$	$\vdots$	$\vdots$

TABLE II
Routing Table RT

routing tables and hence to address nodes, we propose a decentralized beaconing scheme similar in spirit to the centralized scheme in [9]. The aim of the scheme is to build a hierarchical (r, c)-cover, such that every node is

covered at every layer. Additionally, we will build the cover in such a way that a route will exist for every node in Y_j, the set of beacons on level j, to the closest node in Y_{j-1}, for all j (see definition of (r, c)-cover in Section II). We will also make sure that a node only hears a constant, independent of n, number of beacons at every layer. In practice, we will build the hierarchical cover by letting a subset of nodes flood the network. The subset of nodes which floods to build the cover at level j is denoted $beacon(j) = Y_j$. These nodes are the beacons for level j, and $beacon(j) \subset beacon(j+1)$. Further, we denote by $beacon_u$ the set of all beacons heard by a node u. The flooding algorithm is simple. When a node elects itself as a beacon (see next paragraph), it broadcasts to its neighbors an `Address packet` containing its identifier, a hop count initially set to 0 and the level in the hierarchy at which it is beaconing (see Table III). Note that the size of these packets is $O(\log n)$ bits, which is the minimum required to uniquely name every node. These neighbors add an entry to their routing table containing

Packet Type	*Beacon Name*	*Hop Count*	*Level*
$O(1)$ bits	$O(\log n)$ bits	$O(\log n)$ bits	$O(\log \Delta)$ bits

TABLE III
ADDRESS PACKET

the beacon's name, the hop count incremented by 1 and the level of the beacon (and potentially a time stamp). All nodes know, through a timer mechanism explained in the next paragraph, what level is currently being addressed. In turn, these nodes resend the packet after having incremented the hop count to 1. Neighbors of the neighbors will follow the same steps. However, a node only forwards an address packet if it has not received another copy of the packet with the same or a smaller hop count. The latter check ensures that there are no looping packets and that every node only transmits $O(1)$ copies of the packet. Here we make the implicit assumption that the packet travel with approximately the same delay in all directions. Note that the distance stored by the nodes is the shortest path distance to the beacon in the underlying communication graph. The flooding algorithm is shown in Algorithm 1.

Algorithm 1: FLOODING Algorithm: procedure followed by a $u \in V$ node when it receives an `address packet`

Data: Address Packet P_{in}, Level j
Result: Address Packet P_{out} or $\emptyset$/ Updated routing Table
Set $beacon = P_{in}$.(beacon name);
Set $hops = P_{in}$.(distance);
if $beacon \notin RT$.(*beacon name*) **then**
  add $[beacon, hops, phase]$ as a new entry to the RT ;
  $P_{out} = P_{in}$;
  $P_{out}.(distance) = P_{in}.(distance) + 1$;
  broadcast P_{out} to $\mathcal{N}(u)$;
else
  if $hops < RT.(beacon).(distance)$ **then**
    update entry with new hop count;
    $P_{out} = P_{in}$;
    $P_{out}.(distance) = P_{in}.(distance) + 1$;
    broadcast P_{out} to $\mathcal{N}(u)$;
  else
    drop packet
  end
end

The beaconing process is subdivided into phases, one to build each layer of the hierarchy. In practice, nodes should always know what phase is taking place by using a timer mechanism (*i.e.*, one timeslot is allocated to each layer and all nodes have a synchronized clock). We will not enter into the details of scheduling, but note that in an efficient implementation we have spatial diversity in the sense that distant beacons with a small flooding radius

cannot hear each other and work independently. This allows us to shorten the timeslots. In our beaconing scheme, we randomly draw a permutation of the nodes, which will determine the flooding order. In practice, again, this could be implemented with the help of a random backoff mechanism. However, we will stick with the random permutation π terminology in the sequel for the sake of clarity. At any level j (that is when we are in the phase corresponding to level j), nodes will flood in the order dictated by the random permutation. However, a node will be silenced and not elect itself as a beacon in case it is within distance $\frac{\Delta}{c^j}$ of a beacon which has already flooded. We call the cluster of a beacon node u at level j ($cluster_u(j)$) the ball $B_u(\frac{\Delta}{c^j})$. Obviously there can be only one beacon per cluster. As explained above, beacons will flood at last as far as necessary to make sure that they are heard by a beacon at level $j-1$, that is at least $\frac{\Delta}{c^j}(c+1)$ hops. In fact, we will let the beacons at level j flood $f_j = \frac{\Delta}{c^j}(max(\frac{2}{\delta}, c)+1)$ hops, where δ is an arbitrary constant > 1. If the hop count exceeds this value, packets are simply dropped. The beaconing algorithm is described in Algorithm 2. An important property of the flooding radii is that there sum is of the same order as the largest term.

Property 1: The sum S_k of flooding radii $f_{log_c\Delta}$ to f_k is $O(f_k)$

Proof:

$$\begin{aligned} S_k &= \sum_{j=k}^{log_c\Delta} f_j \\ &= \sum_{j=k}^{log_c\Delta} \frac{\Delta}{c^j} \overbrace{(max(\frac{2}{\delta}, c)+1)}^{\wp\text{: constant}} \\ &= \wp \sum_{j=k}^{log_c\Delta} \frac{\Delta}{c^j} \\ &= \Delta\wp \sum_{i=0}^{log_c\Delta-k} \frac{1}{c^{i+k}} \\ &= \wp\frac{\Delta}{c^k} \sum_{i=0}^{log_c\Delta-k} \frac{1}{c^i} \\ &= O(f_k) \end{aligned}$$

■

Algorithm 2: BEACONING Algorithm: called by node u when its turn comes in the random permutation π at level j (can be implemented in practice in a fully distributed way using a random backoff mechanism)

Data: Level j, radius $r_j = \frac{\Delta}{c^j}$
Result: Address Packet P_{out} or $\emptyset$/ Updated routing Table
Let B be the set of all entries in RT with $level = j$;
Let $d_{closest}$ be the distance to the closest beacon in B;
if $d_{closest} < r_j$ **then**
  do nothing
else
  create new address packet P_{out};
  $P_{out}.(distance) = 0$;
  $P_{out}.(name) = u$;
  $P_{out}.(level) = j$;
  broadcast P_{out} to $\mathcal{N}(u)$;
end

1) Control Traffic: To bound the control traffic necessary to address the nodes, we will rely on the α-doubling property of the metric space to show that a node can only hear a constant number of beacons at every layer. We will first show that a ball of radius $2R$ around any node u can only contain at constant number of balls (clusters) of radius R.

Theorem 4 (Random Cover): Let $B(u, 2R)$ be a ball of radius $2R$ in a graph metric (X, d) with doubling constant α. Then, this ball will be covered by at most α^2 balls of radius R if we use the BEACONING algorithm to cover (if $r_j = R$).

Proof: By definition of an α-doubling metric, there must exist an R-cover of a ball or radius $2R$ consisting of at most α balls of radius R. Recursively, there must also exist an $\frac{R}{2}$-cover consisting of α^2 points. In Algorithm 2, when a beacon is elected inside one of these balls of radius $\frac{R}{2}$, it will include it entirely in its cluster when flooding a no other node in that ball will flood. Consequently, at most α^2 balls will be necessary for the cover. ■

One can easily extend this result to the cover of larger balls

Corollary 1: Let B be a ball of radius $R > r$ in an α-doubling metric space (X, d). Then Algorithm 2 will will elect a subset $C \subset X$ of nodes as beacons (if $r_j = r$) with $|C| \leq (\frac{R}{r})^{2log(\alpha)}$.

Proof: Let $R = 2^i r$. Hence, r is doubled $log\frac{R}{r}$ times to obtain R. By Theorem 4, B can be covered by $\alpha^{2log\frac{R}{r}} = (\frac{R}{r})^{2log(\alpha)}$ balls of radius r. ■

Using Corollary 1, we can easily bound the control traffic to build the hierarchical (r, c)-cover with the desired properties

Theorem 5 (Overhead of Random Cover): A hierarchical random cover can be built with a total overhead of

$$O(n \log_c(\Delta)(max\left\{\frac{2}{\delta}c, c\right\} + 1)^{log\beta} \log n) \text{ bits}$$

using Algorithm 2 at all layers (for $j = [log_c\Delta]$), where $\beta \leq \alpha^2$.

Proof: At the highest level, we produce a random $\frac{\Delta}{c}$-cover C. All beacons in the cover C flood the entire network. By Lemma 1, $|C| \leq c^{log\beta}$. Hence, a node u has to transmit at most $c^{log\beta}$ packets of $O(\log n)$ bits. In the j^th phase, beacons flood a radius $f_j := r_j(max\left\{\frac{2c}{\delta}, c\right\} + 1)$, where $r_j = \Delta/c^j$. By Lemma 1, for any node u, $B(u, f_j)$ will be covered by at most $(\frac{f_j}{r_j})^{log\beta} = (max\left\{\frac{2c}{\delta}, c\right\} + 1)^{log\beta}$ balls of radius r_j in a random cover. We conclude that $\forall j \in [1, ..., log_c\Delta]$, the total overhead is bounded by $n(max\left\{\frac{2c}{\delta}, c\right\} + 1)^{log\beta} \log n$ bits. ■

One should point out that in a more careful implementation, every node could start exactly one scoped flood. Indeed, beacons which flooded at level j do not need to re-flood at levels $> j$. Nodes that are close to a beacon which floods at level j could automatically exclude themselves from the flooding at any layer for which they are within r_j of that beacon. A crucial property for routing on top of our beaconing scheme is that a beacon at level j should always flood at least as far as necessary to be heard by a beacon at level $j - 1$ *i.e.*, the beacons at level j should heard by at least a beacon at level $j - 1$.

Property 2: Fix a node u. There must exist, for $j \in \{1, ..., log_c\Delta\}$, a beacon b_j at distance at most $r_j = \frac{\Delta}{c^j}$ and a beacon b_{j-1} at distance at most $r_{j-1} = \frac{\Delta}{c^{j-1}}$ from u by construction. Then, b_{j-1} hears b_j's flood. Additionally, if $r_k \leq D(s, t) \leq r_{k-1}$, the beacon closest to t in $beacon_s \cap beacon_t$ must be at distance at most r_k from t.

Proof: By construction, $f_j := r_j(max\left\{\frac{2c}{\delta}, c\right\} + 1) \geq r_j(c + 1) = \Delta(\frac{1}{c^{j-1}} + \frac{1}{c^j})$. Consequently, $f_j \geq r_j + r_{j-1} \geq D(b_j, b_{j-1})$. We know that there exists a beacon b at level k such that $D(b, t) \leq r_k$. By triangle inequality, $D(s, b) \leq D(t, b) + D(t, s) \leq f_k$ and so the source s must hear that beacon. ■

2) Stretch: In this section we present a greedy forwarding algorithm based on the address system built in at the beginning of Section IV. Node u receives a packet containing v's identifier and its addresses. Recall that the address of a node is a vector containing pairs beacon identifier and distances to those beacons *i.e.*, $V_v = [\{B_1, D(v, B_1)\}, \{B_1, D(v, B_2)\}, ...]$. Note that the length of the vector can be different for every node. To forward a packet, u first builds a set Ψ_{uv} containing all beacons known by both u and v. In turn, it identifies $closest(v)$, the beacon in Ψ_{uv} closest to v. Finally, it forwards the packet to its neighbor on the shortest path to $closest(v)$

Packet Type	*Destination Name*	*Destination Address*	***Data***
$O(1)$ bits	$O(\log n)$ bits	$O(\log \Delta)$ bits	variable length

TABLE IV
DATA PACKET

(see algorithm 3). Note that in order to reduce the size of the packet header, the source node could only include the beacon closest to the destination at every layer. A data packet is shown in Table IV.

Algorithm 3: Greedy Forwarding

Data: v: identifier of destination, V_v: addresses of destination v,
Result: (h) - $h \in \mathcal{N}(u)$: next hop for the packet
Let $\Psi_{uv} = beacons_u \cap beacons_v$;
$closest(v) = min_{l \in S_{uv}} D(l, v)$;
Set $h = RT.(closest(v)).$(next hop);

We now show that the forwarding algorithm is guaranteed to find a path between any source and destination node.

Theorem 6: Given a source s and destination node t, Greedy Forwarding is guaranteed to find a path between s and t.

Proof: Since beacons at the first level flood the entire network, s must have heard at least from a beacon within $\frac{\Delta}{c}$ of t. Hence, t can forward its packet to this beacon. Then, once we reach a beacon at a given level, we are sure that it has heard from a beacon at the next level by Property 2. ■

Next, we show that we can bound the stretch of the path found with Algorithm 3 (*greedy forwarding* algorithm). First, however, we need to prove a useful lemma.

Lemma 1: Consider three points u, v, w in a metric space (X, d). Further, let $D(u, v) \leq k$ and $D(u, w) \geq \frac{2}{\delta}k$, for $k \geq 0$. Then, $D(u, v) + D(v, w) \leq (1 + \delta)D(u, w)$.

Proof: We make use of the triangle inequality and the hypothesis.

$$\begin{aligned} D(u,v) + D(v,w) &\leq D(u,v) + \overbrace{D(u,v) + D(u,w)}^{D(v,w) \leq D(u,v) + D(u,w)} \leq 2k + D(u,w) \\ &\leq 2(\tfrac{\delta}{2}D(u,w)) + D(u,w) = (1+\delta)D(u,w) \end{aligned}$$

■

The routing stretch for this addressing scheme can be bounded as follows.

Theorem 7: Given a source s and destination node d, Greedy Forwarding is guaranteed to find a path between s and d of length at most $(1 + \delta\frac{c^2+2c-1}{c-1})D(s, d)$.

Proof: Assume w.l.o.g that $r_k \leq D(s, d) \leq r_{k-1}$, where $k \in \{1, ..., \lceil log_c \Delta \rceil\}$ is the largest value such that the previous statement holds. Let b_k be a beacon such that $D(t, b_k) \leq \frac{\Delta}{c^k} = r_k$. By property 2, such a beacon must exist. Finally, let x_k be the first node on the path between s and b_k which knows a beacon within $\frac{\Delta}{c^{k+1}}$ of t. By Lemma 1, we know that:

$$D(s, x_k) + D(x_k, d) \leq D(s, b_k) + D(b_k, d) \leq (1 + \delta)D(s, d)$$

where the first inequality follows from triangle inequality. We can apply it since *greedy forwarding* will forward the packet on the shortest path between s and b_k. Once the packet has reached x_k, it will be forwarded in the direction of b_{k+1}. With a similar reasoning as above we can write

$$\begin{aligned} &D(x_k, x_{k+1}) + D(x_{k+1}, d) \quad \leq (1+\delta)D(x_k, d) \\ &D(x_{k+1}, x_{k+2}) + D(x_{k+2}, d) \quad \leq (1+\delta)D(x_{k+1}, d) \\ &\vdots \end{aligned}$$

Summing up all of the above inequalities we obtain:

$$D(s, x_k) + D(x_k, x_{k+1}) + ... + D(x_{log_c \Delta}, d) \leq (1 + \delta)(D(s, d) + D(x_k, d) + D(x_{k+1}, d)...) - (D(x_k, d) + D(x_{k+1}, d) + ...),$$

which leads to path length, P being bounded as,

$$\begin{aligned} &P \leq (1+\delta)D(s,d) + \textstyle\sum_{j=k}^{log_c \Delta} D(x_k, d) \\ &\leq (1+\delta)D(s,d) + \delta \textstyle\sum_{j=k}^{log_c \Delta} max\left\{\frac{2}{\delta}, 1\right\} \frac{\Delta}{c^j}(1 + \frac{1}{c}) \\ &\leq (1+\delta)D(s,d) + \delta max\left\{\frac{2}{\delta}, 1\right\}(1 + \tfrac{1}{c})(\tfrac{\Delta}{c^{k+1}} - 1)(\tfrac{c}{c-1}) \\ &\leq (1+\delta)D(s,d) + \delta max\left\{\frac{2}{\delta}, 1\right\}(\tfrac{c+1}{c-1})cD(s,d) \\ &\leq (1 + \delta\tfrac{c^2+2c-1}{c-1})D(s,d) = O(1)D(s,d) \end{aligned}$$

■

B. Addressed Routing in Dynamic Graphs

The κ-constrained nature of the communication graph $\mathcal{G}^{(t)}$ implies that distances between nodes and beacons can only change by κ from one graph to the next *i.e.,* from one time instant to the next. In other words, a node u can only hear a constant number of beacons on level j at time 0, more precisely those which are inside $B_u(f_j)$. If the beacons elected at time 0 flood f_j hops at every time step, at time step t, node u will hear at most all beacons inside $B_u(f_j + \kappa t)$ at time 0. No other beacon could have moved sufficiently close. This idea is illustrated

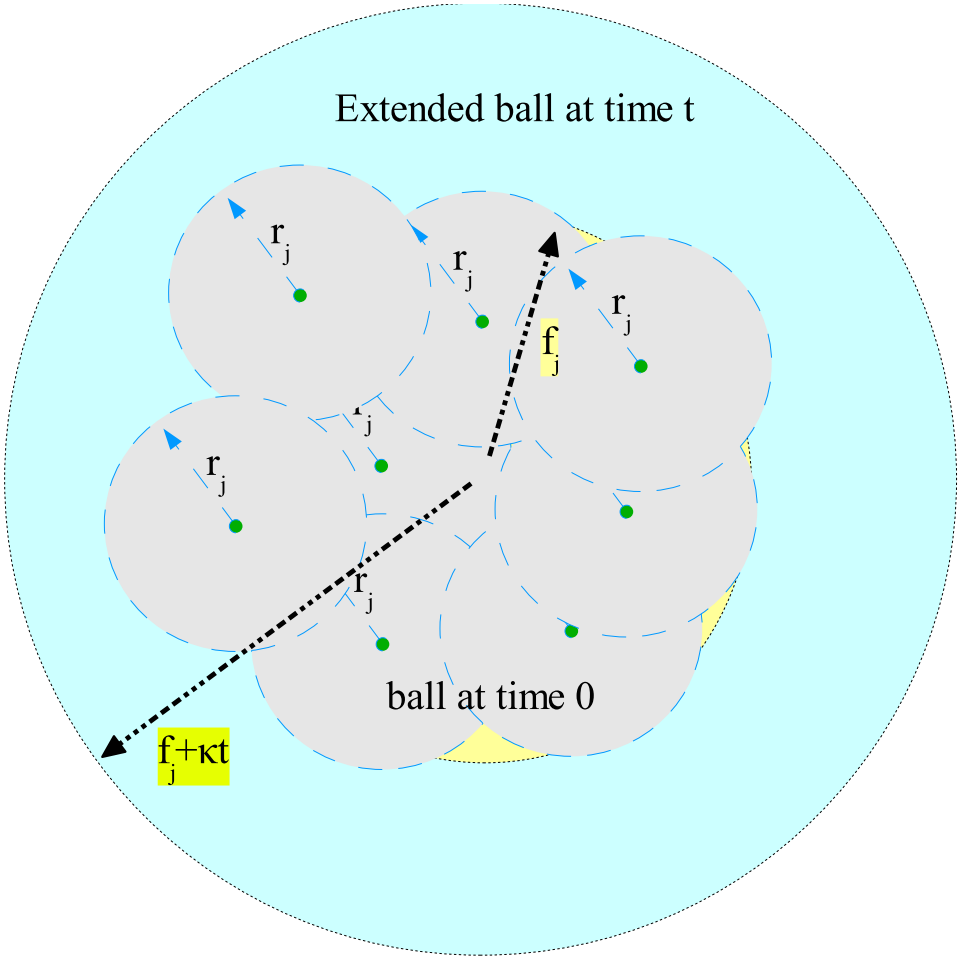

Fig. 1. A node u can only hear a constant number of beacons on level j at time 0, more precisely those which are inside $B_u(f_j)$. If the beacons elected at time 0 flood at every time step, at time step t, node u will hear at most all beacons inside $B_u(f_j + \kappa t)$ at time 0. No other beacon could have moved sufficiently close.

in Figure 1. Of course, if we let all beacons elected at time 0 re-flood the network with `address packets` every time step, the fact that the metric sequence is κ-constrained would not help us gain anything in terms of control traffic overhead, and we could as well apply the algorithm presented in subsection IV-A separately to all graphs. Alternatively, if we simply tried to "repair" the routes to the beacons without re-flooding (*e.g.*, a node starts a local search to find a path to the node through which it routed packets for a particular beacon, if it looses this node as a direct neighbor), the stretch might grow unbounded after some iterations and some node pairs might simply not be able to communicate anymore. However, in practice we would only need to "update" the routes to the beacons. Indeed, we know that from one time instant to the next, the shortest path distance from a node to a beacon can change by at most κ hops. Additionally, only a small set of beacon nodes could have generated update message within a ball of radius $O(f_j)$ hops. This information can be encoded efficiently using `address update packets` (see table V) of constant size. First, a temporary identifier with a constant number of bits m will be

Packet Type	*Color*	*Hop Count* $mod(2\kappa + 1)$
$O(1)$ bits	$O(1)$ bits	$O(1)$ bits

TABLE V
ADDRESS UPDATE PACKET

sufficient for a node to uniquely identify the beacons it has heard from at the time of addressing. Given that a node only knows a constant number of beacons a level j, the problem is similar to a graph coloring problem and a greedy algorithm allows us to solve it in a distributed way. Hence, we shall denote the temporary identifier of

a beacon node b by $\mathcal{H}(b)$, and call it the beacon's color. Second, by computing a hop count $mod(2\kappa+1)$, nodes can uniquely determine their new hop count, given their old hop count.

Another concern, as mentioned above, is the stretch and the existence of routes at any time. Intuitively, we will let the stretch grow at level j by only a constant *i.e.,* the distance between a node and its beacon at level j will at most be multiplied by a constant. Obviously, this process will take more time for the higher layers (small j) than for the lower layers. Nodes will only forward `address update packets` from beacons they heard at the time the level was addressed. Note that these packets do not have a time to live, so they are only dropped when they reach a node which does not know the source beacon. To make sure that communication between all node pairs are possible, we will let beacons flood their address packets $9f_j$ hops, and re-address level j whenever κt is $O(f_j)$. This way, nodes which were within f_j of a beacon at time 0, will always have a route back to that beacon of length at most $2f_j$. Re-addressing of a layer simply amounts to running the algorithm of subsection IV-A from level j downwards. To summarize, we will re-address layer j to $log_c\Delta$ every $\frac{f_j}{\kappa}$ time step, and update that other layers. This process is shown in figure 2.

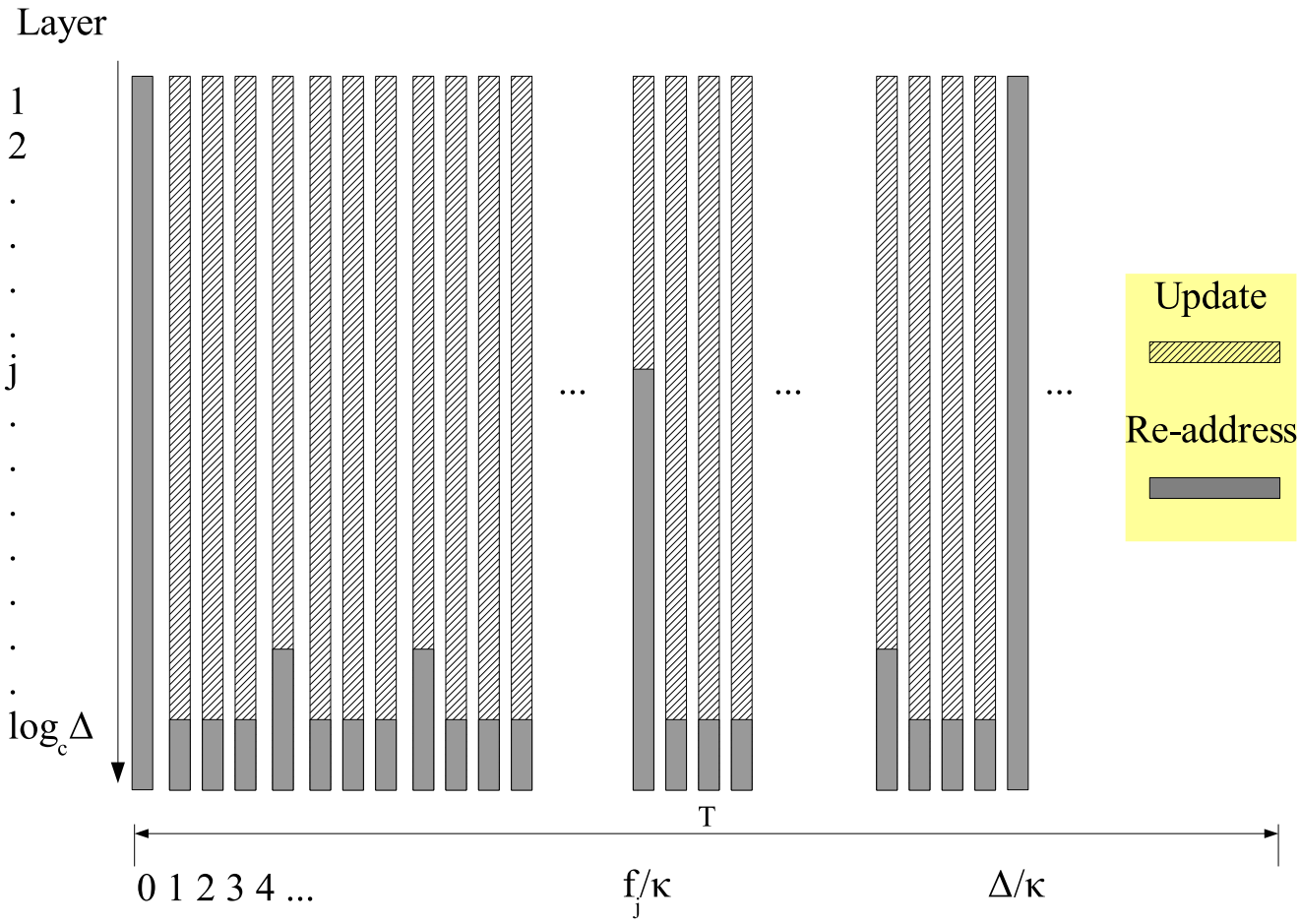

Fig. 2. Layer j is re-addressed every $\frac{f_j}{\kappa}$ time steps. Layers which are not re-addressed are updated using constant size `address update packets`

Once level j has been addressed, beacon nodes run a greedy coloring algorithm, where colors are a number between 1 and 2^m, and m is a constant. If a node has the lowest identifier among the beacons in its $9f_j$ neighborhood which have not yet been colored, it chooses an unassigned color for itself and re-floods its $9f_j$ neighborhood with its color and its identifier. All nodes within $3f_j$ of the beacon will store the color-identifier mapping. Consequently, all beacons flood exactly twice and no beacon has the same color as another beacon within $9f_j$ hops. Note that again a beacon can be surrounded by only a constant number of other beacon in its $9f_j$ neighborhood. The number of colors necessary can consequently be made sufficiently large but is still constant (see Chapter 5 in [10]).

Algorithm 4 is called by all nodes for all $\log_c \Delta$ layers at every time step. Note that before calling the algorithm, if level j is being readdressed in the current time step, a node clears all its routing table entries for all beacons with layers $\geq j$. The flooding algorithm for dynamic beaconing is very similar to Algorithm 1, except that the hop count is $mod(2\kappa + 1)$ and the name of the beacon is replaced by the color thereof.

Algorithm 4: DYNAMIC BEACONING Algorithm: is called by all nodes for all $\log_c \Delta$ layers (here we show the algorithm for 1 particular layer) at every time step

Data: Level j, time step t, radius $r_j = \frac{\Delta}{c^j}$
Result: Address Packet P_{out} or null (no packet to be sent)/ Updated routing Table
Let B be the set of all entries in RT with $level = j$;
Let $d_{closest}$ be the distance to the closest beacon in B;
if $t \ mod(\frac{f_j}{\kappa}) = 0$ **then**
  if $d_{closest} < r_j$ **then**
    do nothing
  else
    create new address packet P_{out};
    $P_{out}.(distance) = 0$;
    $P_{out}.(name) = u$;
    $P_{out}.(level) = j$;
    broadcast P_{out} to $\mathcal{N}(u)$;
  end
else
  if u *is a beacon at level* j **then**
    create new address update packet P_{out};
    $P_{out}.(distance) = 0$;
    $P_{out}.((name)) = \mathcal{H}(u)$;
    broadcast P_{out} to $\mathcal{N}(u)$;
  end
end

1) Control Traffic: In this subsection we will bound the control traffic overhead for addressing nodes in a κ-constrained, α-doubling network, while in the next subsection we will bound the route stretch. To bound the control traffic overhead, we consider a period of T time steps. Level j is re-addressed $\frac{T\kappa}{f_j}$ times during that time period. Simultaneously, we need to consider the control traffic necessary to update the addresses.

Theorem 8 (address update average control traffic overhead): In a κ-constrained, α-doubling graph sequence, the average control traffic overhead for address updating in algorithm 4 is $O(n \log \Delta)$ bits.

Proof: We will be very conservative and consider that all layers are updated at every time step. When layer j is re-addressed, all beacons elected at this layer flood their color $3f_j$ hops. A node can consequently hear at most $(\frac{3f_j}{r_j})^{2log(\alpha)} = (3max(\frac{2}{\delta}, c) + 1)^{2log(\alpha)} \equiv \wp$ (see Corollary 1) beacons at this level. Remember that subsequently, a node only forwards update packets from beacon nodes it heard of at the time of the addressing. The overhead could only be higher if a node forwarded the update packet from a beacon it confuses with another beacon of the same color. However, this cannot happen because two nodes which heard the flood from two different beacons with the same color at time zero were at least $3f_j$ apart (recall that a beacon cannot have another beacon with the same color within $9f_j$, and that only nodes within $3f_j$ know the color-name mapping and forward a beacon's update packets). Before re-addressing, they can move at most $2f_j$ closer to each other, leaving a gap of f_j. Hence, a node will never forward the update packets of a beacon it is not supposed to. Finally, there are $\log_c \Delta$ layers, and for every layer a node transmits at most a constant $\wp$ times a packet of constant size. One can conclude that the average control traffic overhead for address update is upper bounded by $O(n \log \Delta)$ bits. ■

This cost will be dominated by the cost to re-address nodes, which we bound in the following theorem.

Theorem 9 (re-addressing average control traffic overhead): In a κ-constrained, α-doubling graph sequence, the average control traffic overhead for re-addressing in algorithm 4 is $O(n \log n)$ bits.

Proof: When layer j is re-addressed, all beacons elected at this layer flood $9f_j$ hops with `address packets`. These beacons flood at most 2 times (once for addressing and once for coloring). A node can consequently hear at most $2(\frac{9f_j}{r_j})^{2log(\alpha)} = 2(9max(\frac{2}{\delta}, c) + 1)^{2log(\alpha)} \equiv \wp$ (see Corollary 1) beacons. The size of an address packet is $O(\log n)$ bits. Further, level j is re-addressed $\frac{T\kappa}{f_j}$ in a time period of length T time steps. One can now easily compute the total control traffic overhead by summing over all layers:

$$\begin{aligned}\mathcal{T} &\leq n\sum_{j=1}^{\log_c \Delta} \wp \log n \tfrac{T\kappa}{f_j} \\ &= (nT\log n\wp)\sum_{j=1}^{\log_c \Delta} \underbrace{(\frac{1}{f_j})}_{<1} \\ &= O(nT\log n)\end{aligned}$$

which leads to an average control traffic overhead of $O(n\log n)$ bits. ■

Combining Theorems 8 and 9 we obtain the bound on the control traffic in Theorem 2, when we use that fact that $\Delta \leq n$ in an unweighted graph.

2) Stretch: It remains to show that a node which receives an `address update packet` can uniquely determine its current shortest path distance to the beacons it heard from at the time of addressing and that addressed routing with constant stretch between any node pair is possible at any time.

Lemma 2: In a κ-constrained, α-doubling graph sequence, a node u can uniquely determine its shortest path distance to v in $\mathcal{G}^{(t+1)}$ given its shortest path distance to v in $\mathcal{G}^{(t)}$ and the current hop count $mod(2\kappa+1)$.

Proof: Let $d^{(t)}(u,v)$ denote the shortest path distance between nodes u and v at time t. Given that the graph sequence is κ-constrained, we know that $d^{(t+1)}(u,v) \in \left[d^{(t)}(u,v) - \kappa, d^{(t)}(u,v) + \kappa\right]$. This interval is of length $2\kappa+1$. Consequently, only a single multiple of $2\kappa+1$ can fall in this interval. Given the remainder of the division by $2\kappa+1$ *i.e.,* the hop count $mod(2\kappa+1)$, we can uniquely determine the new hop count. ■

Finally, we will show that the stretch is constant

Theorem 10 (Stretch for Addressed Routing in Dynamic Graphs): Algorithm 4 allows $O(1)$ stretch routing in a κ-constrained, α-doubling graph sequence, with a probability which can be made arbitrarily close to 1.

Proof: Consider two nodes u and v such that at time 0, $r_k \leq D(u,v) \leq r_{k-1}$ for some $k \in [\log_c \Delta]$. W.l.o.g we will prove the result for a sequence of graphs starting at time 0, but we could start at any point in time when level k gets re-addressed. At time 0, by theorem 7 the route found by algorithm 3 is such that $S_{uv} = O(1)$. Let us denote by $r^{(t)}(u,v)$ the length of the route between u an v at time t. We will bound the overhead by using the fact the number of beacons is logarithmic in the distance between u and v (to see that, apply property 2 recursively) and that the additive stretch at every layer is bounded since we re-address every layer at a different frequency. Now, one can see that:

$$\begin{aligned}d^{(t)}(u,v) &\leq r^{(t)}(u,v) \\ &\leq r^{(1)}(u,v) + \textstyle\sum_{s=1}^{log_c(D(u,v))} add(s) \\ &\leq O(1)D^{(0)}(u,v) + \sum_{s=1}^{log_c(D(u,v))} add(s) \\ &\qquad\qquad \leq f_{k-1} = O(D^t(u,v)) \\ &\leq O(1)(D^{(t)}(u,v) + \overbrace{t\kappa}) + \textstyle\sum_{s=1}^{log_c(D(u,v))} f_s \\ &\leq O(1)D^{(t)}(u,v)\end{aligned}$$

Where we used the fact the additive distortion for step s from the target ($add(s)$) cannot be larger than f_s, given that level s is re-addressed every $\frac{f_s}{\kappa}$ time steps and that $\sum_{s=1}^{log_c(D(u,v))} f_s = O(D(u,v))$. Note that we have considered the worst case in which the distance between all beacons increases and the distance between the source and the destination decreases. The proof can easily be extended to the other cases. Finally, the routing could fail if the address updating (see Theorem 8) failed. We showed that the probability of this event can be made arbitrarily small. ■

V. Named Routing

In section IV, we have made the assumption that the source node knows the address of the destination and can include it in the packet header. This assumption amounts to having a distance oracle, which could take the form of a central server acting as an "address book" and accessible by all nodes. In practice, especially in fully distributed

environments, this assumption is often unrealistic. In this section, we present a named routing scheme. Even though our scheme relies implicitly on the addresses of the nodes, the latter need never be transmitted and source nodes only need to know the name of the destination. We present two schemes. The first scheme requires a smaller total control traffic overhead but some nodes are overloaded. The second scheme leads to better load-balancing but the total control traffic overhead is higher. We call those two scheme the "super node" and the "load-balanced" scheme respectively.

A. Super Node Scheme (SNS)

Packet Type	*Beacon Name*	*Node Name*
$O(1)$ bits	$O(\log n)$ bits	$O(\log n)$ bits

TABLE VI
ADDRESS PACKET

This scheme is very simple to describe. Whenever a node receives an `address packet` from a beacon b at level j and is in $cluster_b(j)$, it replies to that beacon with its name in a `registration packet` containing the node's and the beacon's names (see Table VI). Beacons then store the identifier of all nodes inside their cluster.

Theorem 11 (Control Traffic Overhead): The average control traffic overhead for SNS in a κ-constrained, α-doubling graph sequence is $O(n \log_c \Delta \log n)$.

Proof: Level j gets addressed every f_j/κ iterations. Hence, in a interval of T time steps, a node u will receive $\frac{T\kappa}{f_j}$ `address packets` from beacons at layer j. For each of these packets, u replies to the beacon which is at distance at most r_j hops. The size of the `registration packet` is $O(logn)$ bits. Consequently, the total control traffic is at most:

$$\begin{aligned}\mathcal{T} &\leq n \sum_{j=1}^{\log_c \Delta} \tfrac{T\kappa}{f_j} r_j \log n \\ &\overbrace{\leq}^{\frac{r_j}{f_j}<1} nT \sum_{j=1}^{\log_c \Delta} \log n \\ &= O(nT \log_c \Delta \log n)\end{aligned}$$

which leads to an average control traffic overhead of $\mathcal{T} = O(n \log_c \Delta \log n)$ bits. ■

Note however that whenever a beacon floods to address nodes, it will get overloaded in the sense that it will receive a `registration packet` for all nodes in its cluster, which could mean $O(n \log n)$ bits for a layer 1 beacon! One way to alleviate this load is to periodically draw a new random permutation, so that over a very long run, every node gets overloaded the same fraction of time. Another issue is the fact that to receive all registrations in one time step, the links around the beacons should have a very high capacity or a time step should "last" very long. The load balanced scheme tries to avoid these caveats.

B. Load-Balanced Scheme LBS

The load balanced scheme is similar to the SNS, except that beacon nodes do not store the identifiers of the nodes in their cluster. Rather, these identifier (names) are distributed among the nodes inside the cluster. In order to distribute the names inside a $cluster_u(j)$, a node sends its identifier toward the beacon v at level $j+1$ with its name closest to u. In turn, when the packets enters $cluster_v(j+1)$, the packet is redirected toward the beacon w at level $j+2$ with its identifier closest to u. The distance between identifiers can simply be computed as the $|u-v|$. We repeat this process until the packet reaches a cluster with a single node. Figure 3 illustrates the LBS.

Theorem 12: The average control traffic overhead for SNS is $O(n \log_c^2 \Delta \log n)$ in a κ-constrained, α-doubling graph sequence.

Proof: Level j gets addressed every f_j/κ iterations. Hence, in a interval of T time steps, a node u will receive $\frac{T\kappa}{f_j}$ `address packets` from beacons at layer j. For each of these packets, u replies by sending a packet toward the beacon at level $j+1$ which is at distance at most $2r_j$ hops. Similarly, once the beacons enters the adequate cluster at level $j+1$, it will travel at most $2r_{j+1}$ hops toward the beacon at level $j+2$. This process repeats at most for j levels. Given that $f_j \geq r_j$, we can apply property 1 to see that the number of hops traversed is $O(f_j)$. The size of the `registration packet` is $O(logn)$ bits.

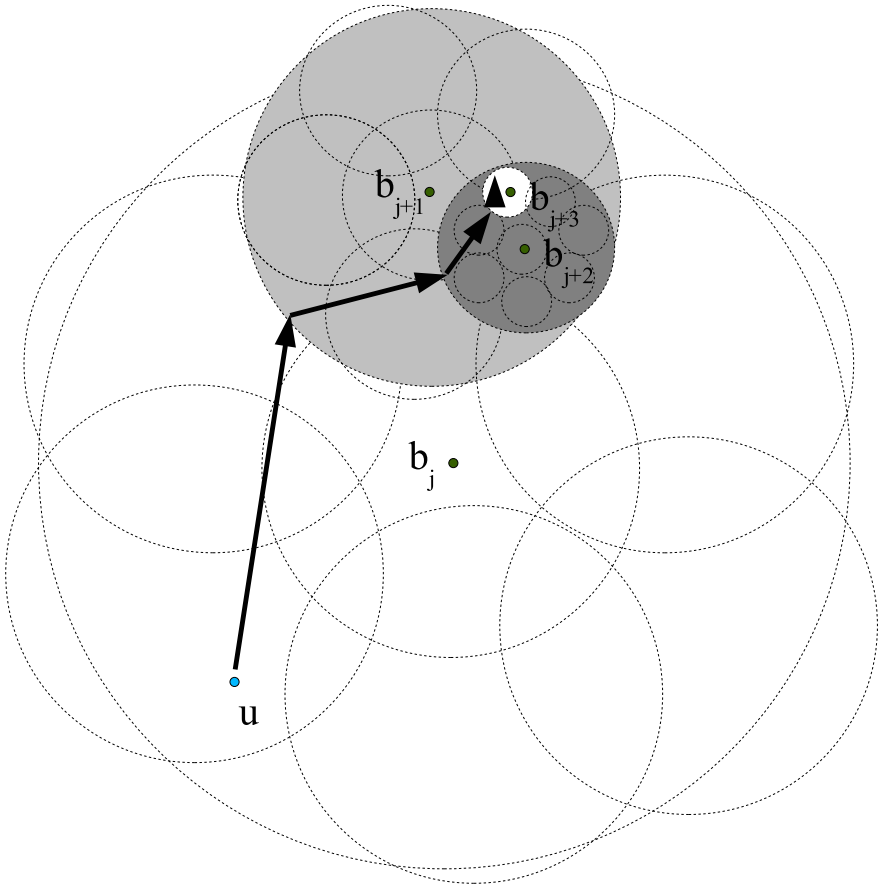

Fig. 3. We show a `registration packet` at level j. The packet first travels toward b_{j+1}, the beacon at level $j+1$ in $cluster_{b_j}(j)$ with the name closest to u, once it enters the cluster, it is redirected toward b_{j+2}, the beacon at level $j+2$ in $cluster_{b_{j+1}}(j+1)$ with the name closest to u. The process continues until a cluster is reached with a single node.

Note that over all layers, a node u sends out its name to $\log_c \Delta$ nodes, which "represent" this node inside a cluster. In order for this representative to be reachable, it must be the node which lies on the path of the beacons with the identifier closest to u. However, whenever level j gets readdressed, this representative must resend the name of u toward the new beacon at level $j+1$ with its identifier closest to u's. The length of the path is again $O(f_j)$ hops. Since there are $\log \Delta$ copies of each node's name, the total control traffic is at most:

$$\begin{aligned}\mathcal{T} &\leq cn \log \Delta \sum_{j=1}^{\log_c \Delta} \frac{T\kappa}{f_j} f_j \log n \\ &\leq cnT \log \Delta \sum_{j=1}^{\log_c \Delta} \log n \\ &= O(nT \log \Delta \log_c \Delta \log n)\end{aligned}$$

where c is a constant. This leads to an average control traffic overhead of $\mathcal{T} = O(n \log_c^2 \Delta \log n)$ bits. ■

Note that with this scheme beacons are neither overloaded in terms of memory requirements nor in terms of amount of traffic to be forwarded. Indeed, the membership publication messages are redirected as soon as the cluster of a given beacon is reached and the identifiers are spread out among the nodes in the cluster.

C. Named Routing

The idea underlying named routing is the same for both the SNS and the LBS. The source node sends a `probe packet`, of the form shown in Table VII. This packet contains the name of the source, the name of the destination as well as the name of the beacons it is aiming for. A constant number of copies of the `probe packet` are first sent toward all beacons of level $\log_c \Delta$ the source node s has in its routing table. Remember that s only knows a

Packet Type	*Beacon Name*	*Destination Name*	*Source Name*	*Membership*
$O(1)$ bits	$O(\log n)$ bits	$O(\log n)$ bits	$O(\log n)$ bits	$O(1)$ bits

TABLE VII
ADDRESS PACKET

constant number of beacons at every layer, and that these beacons are within $O(f_{\log_c \Delta})$ hops. For each copy of the packet, the source node will get the `probe packet` back with the *membership* field set to YES if the destination node is in the cluster of that particular beacon, and NO otherwise. If the destination is not in a cluster at level $\log_c \Delta$, the source will probe all clusters at level $\log_c \Delta - 1$, and so on. Note that the source will ultimately find the destination given that in the worst case it will probe all level 1 clusters and that the destination must belong to one of these. The way the membership to a cluster is tested depends on the scheme.

1) Named Routing with SNS: In the SNS, the packets go all the way to the beacon. Once a packet reaches a beacon, that beacon can immediately answer to the source, given that all nodes in its cluster must have registered with him. If the destination node is not in the cluster, the beacon replies directly to the source after setting the membership field to NO. Nodes on the path between the source and the beacon must temporarily store the reverse path (*i.e.,* the node they received the `probe packet` from) to allow the reverse forwarding. On the other hand, if the destination node is inside the beacon's cluster, that beacon forwards new copies of the `probe packet` to all the lower layer beacons in its cluster. One of these lower layer clusters must contain the destination. That cluster forwards the `probe packet` down the hierarchy again. The other beacons simply drop the packet. Once the packet reaches the destination, that node can answer to the source on the reverse path and the two nodes can now communicate.

Theorem 13: The control traffic to establish a route between a source s and a destination t is $O(1)D(s,t)\log n$ bits and the route is such that S_{uv} is O(1).

Proof: Consider w.l.o.g. that $r_k \geq D(s,t) \geq r_{k+1}$. In that case, the destination *must* belong to the cluster of one of the beacons the source node knows at level k. Consequently, the overhead to probe all the levels down to level k, and the overhead to find the destination in that cluster, could have been at most

$$2c_1 \sum_{j=k}^{\log_c \Delta} f_j \log n + O(f_k \log n) = O(f_k \log n) \text{ bits}$$

where c_1 is a constant by 1. Similarly, to go down the hierarchy we have the same geometric sum and consequently the total overhead is $O(1)D(s,t)\log n$ bits and the path followed by the probe packet is of length $O(f_k) = O(1)D(s,t)$ ■

2) Named Routing with LBS: Routing with LBS works in a similar fashion. The difference is that the `probe packet` is routed toward a beacon b at level j, but once it enters b's cluster $cluster_b j$ *i.e.* it reaches a node in $cluster_b j$, the packet is redirected toward the beacon in $cluster_b j$ at level $j+1$ with its identifier closest to the destination's identifier. Note that when building the hierarchical (r,c)-cover, we made sure that a node in $cluster_b j$ can hear the floods and consequently has a routing table entry for all beacons in $cluster_b j$ at level $j+1$. This procedure is then iteratively repeated until we reach a cluster containing a single node. Again, that node can then reply on the reverse path to the source with a NO if the destination is not in that cluster. If the destination is in that cluster, this node will forward the packet toward all level $j+1$ beacons in $cluster_b j$. Again, all these beacons will be probed for the membership of the destination. Once the packet reaches the destination, that node can answer to the source on the reverse path and the two nodes can now communicate.

Theorem 14: The control traffic to establish a route between a source s and a destination t is $O(1)D(s,t)\log n$ bits and the route is such that S_{uv} is O(1).

Proof: The proof is very similar to the proof of Theorem 13. The difference is that the probing is more expensive in the sense that we can not probe the beacon directly, but rather must go down the hierarchy. Let us consider the probing of a beacon b at distance f_j from the source. The `probe packet` will travel at most f_j to reach the cluster of that beacon. Once inside the cluster, the packet will travel at most f_j hops to reach the cluster at level $j+1$ with the having the name closest to the destination's name. Once inside that cluster, the packet will travel at most f_{j+1} hops to reach the to reach the cluster at level $j+2$ with the having the name closest to the

destination's name. Summing up those terms, we see that the overhead necessary to probe this beacon's cluster is of $O(f_j)$ (see property 1) *i.e.,* of the same order as the overhead necessary to contact the beacon directly. ■

It is important to point out that even thought the communication graph is dynamic, the clusters a node belongs to do not change if the corresponding levels are not re-addressed. Consequently, the "path of beacons" to reach the representative of a node do not change. The fact that the distances between beacons might be multiplied by a constant does not affect the order of magnitude of the bounds we derived.

VI. Conclusion

In this document we have presented a novel practical beaconing scheme to build a hierarchical decomposition of a communication graph with certain properties. We have shown that when we are looking at a sequence of α-doubling, κ-constrained graphs, where both α and κ are constants, we can bound the control traffic for $O(1)$ stretch routing, both for named and addressed scheme. An interesting open question is to find a lower bound on the control traffic, both for dynamic and static environments.

REFERENCES

[1] C. Gavoille, "Routing in distributed networks," *ACM SIGACT News*, pp. 36–52, 2001.

[2] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica, "Geographic routing without location information," in *Proc. ACM Mobicom*, 2003, pp. 96–108.

[3] R. Fonseca, S. Ratnasamy, J. Zhao, C. T. Ee, D. Culler, S. Shenker, and I. a. Stoica, "Beacon-vector routing: Scalable point-to-point routing in wireless sensor networks," in *NSDI*, 2005.

[4] D. Tschopp, S. Diggavi, M. Grossglauser, and J. Widmer, "Robust geo-routing on embeddings of dynamic wireless networks," in *INFOCOM*, Anchorage. USA, 2007.

[5] G. Konjeod, A. Richa, and D. Xia, "Optimal scale-free compact routing in doubling networks," in *SODA*, 2007.

[6] K. Talwar, "Bypassing the embedding: algorithms for low dimensional metrics," in *STOC*, Chicago, IL, USA, 2004, pp. 281 – 290.

[7] T.-H. H. Chan, A. Gupta, B. M. Maggs, and S. Zhou, "On hierarchical routing in doubling metrics," in *SODA*, 2005.

[8] I. Abraham, C. Gavoille, A. V. Goldberg, and D. Malkhi, "Routing in networks with low doubling dimension," in *ICDCS '06*, 2006.

[9] J. Fakcharoenphol, S. Rao, and K. Talwar, "A tight bound on approximating arbitrary metrics by tree metrics," *Journal of Computer and System Sciences*, vol. 69, no. 3, pp. 485–497, 2004.

[10] D. B. West, *Introduction to Graph Theory*, 2nd ed. Prentice Hall, 2001.

[11] M. Penrose, *Random geometric graphs*, ser. Oxford studies in probability. Oxford University Press, 2003.

APPENDIX

Common models used in studies on wireless networks are *Unit Disk Graphs* (UDG) and their random variant *Random Geometric Graphs* (RGG). In these two models, nodes are placed on a square of area R^2. What differentiates the two models is that in the former, nodes are placed deterministically, while in the second model they are placed uniformly at random . The channel model is completely deterministic and nodes are connected if their Euclidean distance is below a threshold distance r, called the communication radius. In mathematical terms, two nodes with positions $x, y \in [0, R]^2$ are connected if and only if $\|x - y\| < r$. We will first show that there exist UDG which are not α-doubling (see Section II for a definition of an α-doubling metric). Then, we will conjecture that RGG are α-doubling with high probability.

Theorem 15: There exists an infinite UDG which is not α-doubling, where α is a constant independent of the number of nodes n

Proof: Consider the graph shown in Figure 4. To show that this graph is not α-doubling, we must show that

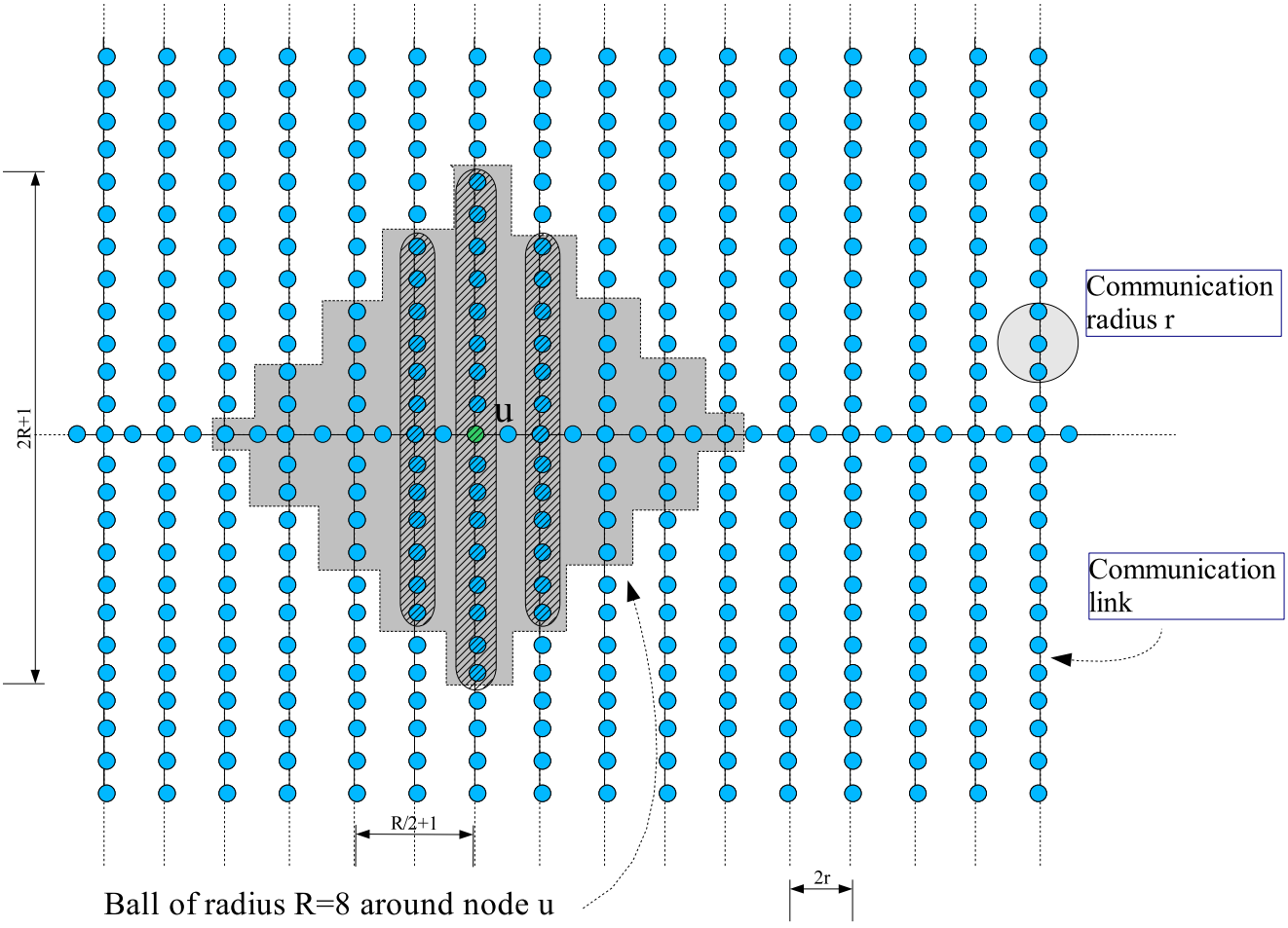

Fig. 4. An infinite UDG obtained by deleting all the nodes in every second column of a grid, except for the nodes on the the middle row. Consequently, "columns" are $2r$ apart.

there exists no constant such that all balls of radius R can be covered a constant α number of balls of radius $R/2$, for all R. Consider the ball centered around u in the figure. One can see that there are $R/4 + 1$ "columns" which cross the middle row at a distance less than $R/2$ from u (that is, the intersection of the column and the row is less than $R/2$ hops away from u). The intersection of each of these columns with $B_u(R)$ is of length more than R (see hatched zones on Figure 4). Consequently, for each of these columns there is at least one node at distance more than $R/2$ from the middle row. To cover these nodes, we need to place at least one ball of radius $R/2$ on each of these columns. Hence, the doubling dimension is lower bounded by $R/4$ and tends to infinity as R goes to infinity. ■

One can notice that in the non-doubling UDG in the proof of Theorem 15 results from a careful construction. We conjecture that when nodes are randomly placed on the unit square $[0,1]^2$ and the communication radius is $O(\frac{1}{\sqrt{n}})$ such that the communication graph is connected w.h.p. (see [11]), such a point constellation is very unlikely to occur and consequently that RGG are α-doubling with high probability.